Validating The Intel® Pentium® 4 Processor

Bob Bentley, Desktop Platforms Group, Intel Corp.
Rand Gray, Desktop Platforms Group, Intel Corp.

Index words: microprocessor, validation, bugs, verification

ABSTRACT

Developing a new leading-edge Intel® Architecture microprocessor is an immensely complicated undertaking. The microarchitecture of the Pentium® 4 processor is significantly more complex than any previous Intel Architecture microprocessor, so the challenge of validating the logical correctness of the design in a timely fashion was indeed a daunting one. In order to meet this challenge, we applied a number of innovative tools and methodologies, which enabled us to keep functional validation off the critical path to tapeout while meeting our goal of ensuring that first silicon was functional enough to boot operating systems and run applications. This in turn enabled the post-silicon validation teams to quickly "peel the onion", resulting in an elapsed time of only ten months from initial tapeout to production shipment qualification, an Intel record for a new IA-32 microarchitecture.

This paper describes how we went about the task of validating the Pentium 4 processor and what we found along the way. We hope that other microprocessor designers and validators will be able to benefit from our experience and insights. As Doug Clark has remarked "Finding a bug should be a cause for celebration. Each discovery is a small victory; each marks an incremental improvement in the design." [1]

INTRODUCTION

The Pentium 4 processor is Intel's most advanced IA-32 microprocessor, incorporating a host of new microarchitectural features including a 400MHz system bus, hyper pipelined technology, advanced dynamic execution, rapid execution engine, advanced transfer cache, execution trace cache, and Streaming Single Instruction, Multiple Data (SIMD) Extensions 2 (SSE2).

PRE-SILICON VALIDATION CHALLENGES AND ISSUES

The first thing that we had to do was build a validation team. Fortunately, we had a nucleus of people who had worked on the Pentium® Pro processor and who could do the initial planning for the Pentium 4 processor while at the same time working with the architects and designers who were refining the basic microarchitectural concepts. However, it was clear that a larger team would be needed, so we mounted an extensive recruitment campaign focused mostly on new college graduates. Not only did this take a large amount of effort from the original core team (at one stage we were spending an aggregate 25% of our total effort on recruiting!), but it also meant that we faced the monumental task of training these new team members. However, this investment paid off handsomely over the next few years as the team matured into a highly effective bug-finding machine that found more than 60% of all the logic bugs that were filed prior to tapeout. In doing so, they developed an in-depth knowledge of the Pentium 4 processor's NetBurst™ microarchitecture that has proved to be invaluable in post-silicon logic and speedpath debug and also in fault grade test writing.

For the most part, we applied the same or similar tools and methodologies that were used on the Pentium Pro processor to validate the Pentium 4 processor. However, we did develop new methodologies and tools in response to lessons learnt from previous projects and also to address some new challenges raised by the Pentium 4 processor design. In particular, the use of Formal Verification, Cluster Test Environments, and focused Power Reduction Validation was either new or a greatly extended form than that used on previous projects. These methodologies and tools are discussed in detail in later sections of this paper.

Pre-Silicon Validation Environment

Except for Formal Verification (FV), all pre-silicon validation was done using either a cluster-level or full-chip SRTL model running in the CSIM simulation environment developed by Intel Design Technology. We ran these simulation models on either interactive workstations or compute servers. Initially, these were legacy IBM RS/6000s* running AIX*, but over the course of the project we switched to systems based on the Pentium® III processor, running Linux*. Our computing pool grew to encompass several thousand systems by the end of the project, most of them compute servers. We used an internal tool called Netbatch to submit large numbers of batch simulations to these systems, which we

were able to keep utilized at over 90% of their maximum 24/7 capacity. By tapeout we were averaging 5-6 billion cycles per week and had accumulated over 200 billion (to be precise, $2.384 * 10^{11}$) SRTL simulation cycles of all types.

Formal Verification

The Pentium 4 processor was the first project of its kind at Intel to apply FV on a large scale. We decided early in the project that the FV field had matured to the point where we could consider trying to use it as an integral part of the design verification process rather than only applying it retroactively, as had been done on previous products such as the Pentium Pro processor. However, it was clear from the start that we couldn't formally verify the entire design—that was (and still is) way beyond the state of the art for today's tools. We therefore decided to focus on the areas of the design where we believed FV could make a significant contribution; in particular, we focused on the floating-point execution units and the instruction decode logic. Because these areas had been sources of bugs in the past that escaped early detection, using FV allowed us to apply this technology to some real problems with real payback.

One of the major challenges for the FV team was to develop the tools and methodology needed to handle a large number of proofs in a highly dynamic environment. For the most part we took a model-checking approach to FV, using the Prover tool from Intel's Design Technology group to compare SRTL against separate specifications written in Formal Specification Language (FSL). By the time we taped out we had over 10,000 of these proofs in our proof database, each of which had to be maintained and regressed as the SRTL changed over the life of the project. Along the way, we found over 100 logic bugs—not a large number in the overall scheme of things, but 20 of them were "high-quality" bugs that we believe would not have been found by any of our other pre-silicon validation activities. Two of these bugs were classic floating-point data space problems:

1. The FADD instruction had a bug where, for a specific combination of source operands, the 72-bit FP adder was setting the carryout bit to 1 when there was no actual carryout.
2. The FMUL instruction had a bug where, when the rounding mode was set to "round up", the sticky bit was not set correctly for certain combinations of source operand mantissa values, specifically:

   src1[67:0] := $X*2^{(i+15)} + 1*2^{i}$

   src2[67:0] := $Y*2^{(j+15)} + 1*2^{j}$

   where i+j = 54, and {X,Y} are any integers that fit in the 68-bit range.

Either of these bugs could easily have gone undetected[1], not just in the pre-silicon environment but also in post-silicon testing.

We put a lot of effort into making the regression of the FV proof database as push-button as possible, not only to simplify the task of running regressions against a moving SRTL target but because we viewed reuse as being one of the keys to proliferating the quality of the original design. This approach has had an immediate payoff: a regression of the database of 10,000 proofs on an early SRTL model of a proliferation of the Pentium 4 processor yielded a complex floating-point bug.

Cluster-Level Testing

One of the fundamental decisions that we took early in the Pentium 4 processor development program was to develop Cluster Test Environments (CTEs) and maintain them for the life of the project. There is a CTE for each of the six clusters into which the Pentium 4 processor design is logically subdivided (actually, microcode can be considered to be a seventh logical cluster, and it too has a test environment equivalent to the other CTEs). These CTEs are groupings of logically related units (e.g., all the execution units of the machine constitute one CTE) surrounded by code that emulates the interfaces to adjacent units outside of the cluster and provides an environment for creating and running tests and checking results.

These CTEs took a good deal of effort to develop and maintain, and were themselves a source of a significant number of bugs. However, they provided a number of key advantages:

First and foremost, they provided **controllability** that was otherwise lacking at the full-chip level. An out of order, speculative execution engine like the Pentium® Pro or Pentium 4 processor is inherently difficult to control at the instruction set architecture level. Assembly-language instructions (macroinstructions) are broken down by the machine into sequences of microinstructions that may be executed in any order (subject to data dependencies) relative to one another and to microinstructions from other preceding or following macroinstructions. Trying to produce precise microarchitectural behavior from macroinstruction sequences has been likened to pushing on a piece of string. This problem is particularly acute for the back end of the machine, the memory and bus clusters that lie beyond the out-of-order section of the microarchitecture pipeline. CTEs allowed us to provoke specific microarchitectural behavior on demand.

Second, CTEs allowed us to make significant strides in **early validation** of the Pentium 4 processor SRTL even

[1] We calculated that the probability of hitting the FMUL condition with purely random operands was approximately 1 in $5*10^{20}$, or 1 in 500 million trillion!

before a full-chip model was available. As described below, integrating and debugging all the logic and microcode needed to produce even a minimally functional full-chip model was a major undertaking; it took more than six months from the time we started until we had a "mostly functional" IA-32 machine that we could start to target for aggressive full-chip testing. Because we had the CTEs, we were able to start testing as soon as there was released code in a particular unit, long before we could have even tried to exercise it at the full-chip level.

Even after we had a full-chip model, the CTEs essentially **decoupled validation** of individual unit features from the health of the full-chip model. A blocking bug in, for example, the front end of the machine did not prevent us from continuing to validate in other areas. In actual fact, we rarely encountered this kind of blockage because our development methodology required that all changes be released at cluster level first, and only when they had been validated there did we propagate them to the full-chip level. Even then, we required that all full-chip model builds pass a mini-regression test suite before they could be released to the general population. This caught most major cross-unit failures that could not be detected at the CTE level.

One measure of the success of the CTEs is that they caught almost 60% of the bugs found by dynamic testing at the SRTL level. Another is that, unlike the Pentium Pro processor and some other new microarchitecture developments, the Pentium 4 processor never needed an SRTL "get-well plan" at the full-chip level where new development is halted until the health of the full-chip model can be stabilized.

POWER REDUCTION VALIDATION

From the earliest days of the Pentium 4 processor design, power consumption was a concern. Even with the lower operating voltages offered by P858, it was clear that at the operating frequencies we were targeting we would have difficulty staying within the "thermal envelope" that was needed to prevent a desktop system from requiring exotic and expensive cooling technology. This led us to include two main mechanisms for active power reduction in the design: *clock gating* and *thermal management*. Each of these is discussed in other papers in this issue of the *Intel Technology Journal*. Each presented validation challenges—in particular, clock gating.

Clock gating as a concept is not new: previous designs have attempted to power down discrete structures such as caches when there were no accesses pending. What was different about the Pentium 4 processor design was the extent to which clock gating was used. Every unit on the chip had a power reduction plan, and almost every Functional Unit Block (FUB) contained clock gating logic. In all, there were several hundred unique clock gating conditions identified, and each one of them needed to be validated from several different perspectives:

1. We needed to verify that each condition was implemented as per plan and that it functioned as originally intended. We needed to verify this not once, but continually throughout the development of the Pentium 4 processor, as otherwise it was possible for power savings to be eroded over time as an unintended side effect of other bug or speedpath fixes. We tackled this problem by constructing a master list of all the planned clock-gating features, and writing checkers in *proto* for each condition to tell us if the condition had occurred and to make sure that the power down had occurred when it should have. We ran these checkers on cluster regressions and low-power tests to develop baseline coverage, and then wrote additional tests as necessary to hit uncovered conditions.
2. While establishing this coverage, we had to make sure that the clock-gating conditions did not themselves introduce new logic bugs into the design. It is not hard to imagine all sorts of nightmare scenarios: unit A is late returning data to unit B because part of A was clock gated, or unit C samples a signal from unit D that is undriven because of clock gating, or other variations on this theme. In fact, we found many such bugs, mostly as a result of (unit-level) design validation or full-chip microarchitecture validation, using the standard set of checkers that we employed to catch such implementation-level errors. We had the ability to override clock gating either selectively or globally, and we developed a random power down application programming intereface (API) that could be used by any of the validation teams to piggyback clock gating on top of their regular testing. Once we had developed confidence that the mechanism was fundamentally sound, we built all our SRTL models to have clock gating enabled by default.
3. Once we had implemented all the planned clock-gating conditions, and verified that they were functioning correctly, we relied primarily on measures of clock activity to make sure that we didn't lose our hard-won power savings. We used a special set of tests that attempted to power down as much of each cluster as possible, and collected data to see what percentage of the time each clock in the machine was toggling. We did this at the cluster level and at the full-chip level. We investigated any appreciable increase in clock activity from model to model, and made sure that it was explainable and not due to designer error.
4. Last, but by no means least, we tried to make sure that the design was cycle-for-cycle equivalent with clock gating enabled and disabled. We had established this as a project requirement, to lessen the likelihood of undetected logic bugs or performance degradation caused by clock gating. To do this, we developed a methodology for *temporal divergence*

testing, which essentially ran the same set of tests twice, with clock gating enabled and disabled, and compared the results on a cycle-by-cycle basis.

We organized a dedicated Power Validation team to focus exclusively on this task, and they filed numerous bugs as a result of their pre-silicon validation (we filed "power bugs" whenever the design did not implement a power-saving feature correctly, whether or not it resulted in a functional failure). The results exceeded our fondest expectations: not only was clock gating fully functional on A-0 silicon, but we were able to measure approximately 20W of power saving in a system running typical workloads.

Full-chip Integration and Testing

With a design as complex as the Pentium 4 processor, integrating the pieces of SRTL code together to get a functioning full-chip model (let alone one capable of executing IA-32 code) is not a trivial task. We developed an elaborate staging plan that detailed just what features were to be available in each stage and phased in this plan over a 12-month period. The Architecture Validation (AV) team took the lead in developing tests that would exercise the new features as they became available in each phase, but did not depend upon any as-yet unimplemented IA-32 features. These tests were throwaway work—their main purpose was to drive the integration effort, not to find bugs. Along with these tests we developed a methodology which we called *feature pioneering*: when a new feature was released to full-chip for the first time, a validator took responsibility for running his or her feature exercise tests, debugging the failures, and working with designers to rapidly drive fixes into graft (experimental) models, thereby bypassing the normal code turn-in procedure, until an acceptable level of stability was achieved. Only then was the feature made available for more widespread use by other validators. We found that this methodology greatly speeded up the integration process. It also had a side effect: it helped the AV team develop their full-chip debugging skills much more rapidly than might otherwise have occurred.

Once a fully functional full-chip SRTL model was available, these feature pioneering tests were discarded and replaced by a new suite of IA-32 tests developed by the AV team, whose purpose was to fully explore the architecture space. Previous projects up to and including the Pentium Pro processor had relied on an "ancestral" test base inherited from past projects, but these tests had little or no documentation, unknown coverage, and doubtful quality (in fact, many of them turned out to be bug tests from previous implementations that had little architectural value). We did eventually run the "ancestral" suite as a late cross-check, after the new suite had been run and the resulting bugs fixed, but we found nothing of consequence as a result.

Coverage-Based Validation

Recognizing the truth of the saying: "If it isn't tested, it doesn't work" we attempted wherever possible to use coverage data to provide feedback on the effectiveness of our tests and tell us what we had and had not tested. This in turn helped direct future testing towards the uncovered areas. Since we relied very heavily on direct random test generators for most of our microarchitectural testing, coverage feedback was an absolute necessity if we were to avoid "spinning our wheels" and testing the same areas over and over again while leaving others completely untouched. In fact, we used the tuple of cycles run, coverage gained, and bugs found as our first-order gauge of the health of the SRTL model and its readiness for tapeout.

Our primary coverage tool was Proto from Intel Design Technology, which we used to create coverage monitors and measure coverage for a large number of microarchitecture conditions. By tapeout we were tracking almost 2.5 million unit-level conditions and more than 250,000 inter-unit conditions, and we succeeded in hitting almost 90% of the former and 75% of the latter. For the conditions that we were unable to hit prior to tapeout, we made sure that they were scattered throughout the entire coverage space and not concentrated in a few areas; and we also made sure that the System Validation (SV) team targeted these areas in their post-silicon validation plans. We also used Proto to instrument several thousand multiprocessor memory coherency conditions (combinations of microarchitecture states for caches, load and store buffers, etc.), and, as mentioned above, all the clock-gating conditions that had been identified in the unit power reduction plans. We used the Pathfinder tool from Intel's Central Validation Capabilities group to measure how well we were exercising all the possible microcode paths in the machine. Much to our surprise, running all of the AV test suite yielded coverage of less than 10%; further analysis revealed that many of the untouched paths involved memory-related faults (e.g., page fault) or assists (e.g., A/D bit assist). This made sense, as the test writers had (reasonably enough) set up their page tables and descriptors so as to avoid these time-consuming functions (at SRTL simulation speeds, every little bit helps!), but it did reinforce the value of collecting coverage feedback and not just assuming that our tests were hitting specified conditions.

POST-SILICON VALIDATION

As soon as the A-0 silicon was available, validation moved into the "post-silicon" phase. In post-silicon validation, the processor is tested in a system setting. Validation in this setting concentrates not only on the processor, but its interaction with the chipset, memory system, and peripherals. In this environment, the testing is done at real-time processor speeds, unlike the simulation environment that must be used prior to the

arrival of the actual silicon. This is good news for test coverage, as much longer and more complex tests can be run in real-time, but it is bad news for debugging. In the SRTL simulator, all of the internal signals of the processor are available for inspection, but in the actual silicon, the primary visibility is from the transactions on the processor system bus. A significant effort went into preparing for the availability of the A-0 processor silicon. Hardware engineering teams developed and constructed validation platforms to provide execution and test vehicles for the processor silicon. The SV team assigned engineers to learn the microarchitecture of the processor and develop specific tests for the silicon. The Compatibility Validation (CV) team constructed an elaborate test infrastructure for testing the processor silicon using industry-standard operating systems, application software, and popular peripheral cards. The Circuit Marginality Validation (CMV) team prepared a test infrastructure for correlating tester schmoo plots with actual operational characteristics in systems capable of running standard operating systems as well as SV tests. All of these preparations were completed prior to the actual receipt of the A-0 processor silicon such that testing could proceed without delay as soon as the first silicon samples arrived.

Arrival of First Silicon

Systems developed for post-silicon validation included uniprocessor desktop systems, dual-processor workstation boards, 4MP server boards, and 4MP system validation platforms that include extensive test assistance circuitry (although the Pentium 4 processor is a uniprocessor product, we have found that certain types of *multiprocessor* testing can be good at exposing *uniprocessor* bugs). A few of each of these systems were available in the Pentium 4 processor system validation lab a few weeks prior to the arrival of first silicon samples. Within a few days after receiving the initial samples of A-0 processor silicon, we had successfully booted a number of operating systems including MS- DOS*, MS Windows* 98, MS Windows NT* 4.0, and Linux*.

The most complex and flexible validation platform was the 4MP system validation platform. This platform included the following key features:

- logic analyzer connectors
- SV hooks card that permits direct stimulus of FSB signals
- a software-controllable clock board that permits setting the processor system bus frequency in 1MHz steps
- software-controllable voltage regulators for both the CPU and chipset components
- built-in connectors for the In-Target Probe (ITP) debugging port
- four PCI hublink boards to support a large number of synthetic I/O agents
- sockets for the processor and chipset component silicon

System Validation

In parallel with the hardware system design, a team of System Validation (SV) engineers was assembled from a small core of experienced system validators. Learning from previous SV experiences, the team was assembled early to provide sufficient time for the engineers to learn the microarchitecture of the Pentium 4 processor and to develop an effective test suite. The team was also chartered with improving the effectiveness of system validation. A variety of test strategies, environments, and tools were developed to meet the challenge of accelerating the post-silicon validation schedule while achieving the same or a higher level of test coverage. The SV organization comprised a number of teams that targeted major CPU attributes:

- architecture—including the Instruction Set Architecture (ISA), floating-point unit, data space, and virtual memory
- microarchitecture—focusing on boundary conditions between microarchitectural units
- multi-processor—focusing on memory coherency, consistency, and synchronization

Different test methodologies were developed to test the various processor attributes. SV methodologies and test environments include

- Random Instruction Test (RIT) generators, which are highly effective for ISA testing, especially the interactions between series of instructions
- directed (focused) tests
- directed random tests (algorithmic tests with random code blocks inserted strategically)
- data space (or data domain) tests for testing boundary and random floating-point operands

SV tests are developed to run directly on the processor without an operating system run-time environment. Due to this, and the fact that the full test source is available and understood by the team, SV tests are relatively straightforward to debug in the system environment.

Random Instruction Testing

An especially effective method for testing the interactions between sequences of instructions is the Random

*Other brands and names are the property of their respective owners.

Instruction Test (RIT) environment. It is not mathematically feasible to even test just the possible combinations of a single instruction with all possible operands and processor states. Add to this the possibility of virtually limitless combinations of instruction sequences and it becomes clear that a systematic and exhaustive test strategy is wholly impractical. A practical and effective alternative is to construct an RIT environment. An RIT environment works in this way:

- The RIT tool generates a highly random sequence of instructions, sometimes described as a series of instructions that no sane programmer would ever devise.
- The instructions are presented in sequence to an architectural simulator, which constructs a memory image of the processor state (whenever memory is affected by a store, a push, or an exception frame).
- Once the test generation is concluded, the resulting test object and memory image are saved.
- The test is loaded onto the SV platform and executed.
- Following the conclusion of the test, the SV platform memory is compared against the memory image obtained from the architectural simulator. If the images match, the test passes; otherwise it fails. Another failure possibility is a "hang;" for example, the processor may experience a livelock condition and fail to complete the test.

A number of key requirements drove the development of such a test environment for the Pentium 4 processor:

- The first one was the ability to fully warm up the very long Pentium 4 processor pipeline. This is particularly difficult in an RIT environment, as truly random instruction combinations tend to cause frequent exceptions or other control-flow discontinuities. Typical RIT tools available before the new tool was developed would typically encounter a pipeline hazard within 3 to 20 instructions on average. This could result in missing bugs that might exist in actual application or operating system software.
- The second one was the ability to avoid "false" failures, e.g., failures occurring due to undefined processor states or other differences between an architectural simulator and the actual processor silicon. This is an extremely important feature, as a high false failure rate will limit the useful throughput of such a tool. Every failure must be examined, whether false or real (otherwise, how does one know if a real failure has occurred?).
- The third one was the ability of the RIT environment to fully propagate the processor state to the memory image file without unduly affecting the randomness of the instruction stream. Lacking this feature it is possible to miss failures due to the tendency of RIT tools to frequently overwrite state, thus potentially hiding the failing signature of a bug.
- The fourth one was the ability of the tool to greatly increase RIT throughput. The new tool increased the throughput by a factor of 100 over existing tools. This was essential to find rarely occurring or highly intermittent bugs.

The new tool, known as Pipe-X (for Pipeline Exerciser) proved to be extremely effective, logging the most bugs of any SV test environment or test suite. It has effectively a zero false failure rate, without which the high throughput would prove to be unmanageable from a debugging standpoint. For a given processor stepping that requires production qualification, one billion tests (each 10,000 instructions in length) are executed in approximately eight weeks. To date, approximately 10 billion RIT tests have been executed on the Pentium 4 processor, compared with the approximately 10 million RIT tests that have been executed on all versions of the Pentium® II and Pentium® III processor families. Pipe-X has been found to be effective in finding both architectural and microarchitectural logic bugs.

Focused SV Testing

We used directed or focused testing to complement RIT. It is important to perform algorithmic testing of major processor components. A comprehensive set of focused tests was available from the Pentium Pro, Pentium II, and Pentium III processor families. This test suite is known as the Focus Test Suite (FTS) and is particularly effective at finding cache bugs, Programmable Interrupt Controller (PIC) bugs, and general functional bugs. The focus test suite has been in continuous development for many years, and was effectively doubled in size to prepare for Pentium 4 processor validation. It has found almost the same number of bugs as Pipe-X.

Compatibility Validation

Although SV finds most post-silicon bugs (approximately 71%), and those bugs are the most straightforward to debug, it is vital to ensure that the new processor, chipset, and memory system works correctly with standard operating systems using a wide variety of software applications and peripheral cards. For this reason, an extremely elaborate Compatibility Validation (CV) laboratory was assembled. CV tests are designed to particularly stress interactions between the processor and chipset, concentrating on causing high levels of FSB traffic. The CV staff often work closely with Original Equipment Manufacturers (OEMs) to resolve problems sighted at the OEM and assist in performance validation by running standard benchmark workloads. CV tests also help to weed out software problems in BIOS. The CV team will see most of the bugs that the SV team uncovers, but due to the difficult nature of debugging in the CV

environment, the bugs found in SV are resolved much more quickly.

Debugging in the System Environment

Debugging in the laboratory using actual Pentium 4 processor silicon installed in a validation platform or PC-like test vehicle is difficult in the extreme. The very best place to debug a processor bug is in the processor simulator, where all signals are available for scrutiny. Unfortunately, in the system environment almost no internal signals are visible to the system debugger. A suite of tools was developed for use in the Pentium 4 processor, using both architectural and microarchitectural features of the processor:

The In-Target Probe (ITP) consists of a scan-chain interface with the processor that connects to a host PC system. Using the ITP, the debugger can set breakpoints, examine memory and CPU registers, and start and stop processor program execution. This tool is helpful for interactive patching of test programs, for single-stepping program execution, and for loading small test fragments. In other words, for blatant functional bugs in the processor this tool is effective. However, many bugs only happen when the processor is running at full speed with multiple processors and threads executing, and frequently a bug will immediately disappear when inserting a breakpoint near the failure point.

Scan chain-based array dumps can be used to construct limited watch windows of a small set of select internal signals. This can be especially useful for identifying the signature of some bugs.

Logic analyzer trace captures of the processor system bus can be translated into code streams that can be run on a hardware-accelerated processor RTL model. Such a methodology is based upon forcing all instructions to be fetched on the bus due to periodic invalidation of processor caches.

Validation platform features permit the schmooing of voltage, temperature, and frequency to help accelerate the occurrence of circuit bugs. However, the most effective environment for debugging circuit problems is the semiconductor tester lab.

Due to the extremely complex and lengthy Pentium 4 processor pipeline, many bugs are extremely difficult to reproduce. Being able to capture such failures on a logic analyzer and subsequently running the resulting program fragment on a hardware-accelerated RTL model has time and again proven to be almost the only method for isolating highly intermittent bugs.

Debugging has historically been the primary limiter to post-silicon validation throughput, and despite significant improvements in debugging based on the use of the logic analyzer, it is usually on the critical path to production qualification.

BUG DISCUSSION

Comparing the development of the Pentium® 4 processor with the Pentium® Pro processor, there was a 350% increase in the number of bugs filed against SRTL prior to tapeout. The breakdown of bugs by cluster was also different: on the Pentium Pro processor [2] microcode was the largest single source of bugs, accounting for over 30% of the total, whereas on the Pentium 4 processor, microcode accounted for less than 14% of the bugs. For both designs, the Memory Cluster was the largest source of hardware bugs, accounting for around 25% of the total in both cases. This is consistent with data from other projects, and it indicates that for future projects we should continue to focus on preventing bugs in this area. We also determined that almost 20% of all the bugs filed prior to tapeout were found by code inspection.

We did a statistical study [3] to try to determine how the bugs came to be in the Pentium 4 processor design, so that we could improve our processes for preventing bugs from getting into future designs. The major categories were as follows:

- RTL Coding (18.1%)—These were things like typos, cut and paste errors, incorrect assertions (instrumentation) in the SRTL code, or the designer misunderstood what he/she was supposed to implement.
- Microarchitecture (25.1%)—This covered several categories: problems in the microarchitecture definition, architects not communicating their expectations clearly to designers, and incorrect documentation of algorithms, protocols, etc.
- Logic/Microcode Changes (18.4%)—These were bugs that occurred because: the design was changed, usually to fix bugs or timing problems, or state was not properly cleared or initialized at reset, or these were bugs related to clock gating.
- Architecture (2.8%)—Certain features were not defined until late in the project. This led to shoehorning them into working functionality.

Post-Silicon Bugs

An examination of where bugs have been found in the post-silicon environment reveals the following data:

- System Validation (71%)—The dominance of SV is intentional, as it is definitely the best environment in which to debug post-silicon bugs. A wide spectrum of logic and circuit bugs is found in this environment.
- Compatibility Validation (7%)—Although this team doesn't find as many bugs as SV, the bugs found are in systems running real-world operating systems and applications. Most of the bugs found in SV will also be seen in the CV environment.

- Debug Tools Team (6%)—The preponderance of bugs found by the debug tools team were found in the first few weeks after A-0 processor silicon arrived. This stems from the fact that getting the debug tools working is one of the earliest priorities of silicon validation.
- Chipset Validation (5%)—The chipset validation teams concentrate on bus and I/O operations, and the bugs found by this team reflect that focus: they are typically related to bus protocol problems.
- Processor Architecture Team (4%)—The processor architecture team spends much time in the laboratory once silicon arrives, examining processor performance and general system testing. This team also plays a central role in debugging problems discovered by the SV, CV, and other validation teams.
- Platform Design Teams and Others (7%)—This group includes the hardware design teams that develop and deploy validation and reference platforms, the processor design team, the pre-silicon validation team, and software enabling teams.

CONCLUSION

The Pentium® 4 processor was highly functional on A-0 silicon and received production qualification in only ten months from tapeout. The work described here is a major reason why we were able to maintain such a tight schedule and enable Intel to realize early revenue from the Pentium 4 processor in today's highly competitive marketplace.

ACKNOWLEDGMENTS

The results described in this paper are the work of many people over a multi-year period. It is impossible to list here the names of everyone who contributed to this effort, but they deserve all the credit for the success of the Pentium® 4 processor validation program.

REFERENCES

[1] Clark, D, "Bugs Are Good: A Problem-Oriented Approach To The Management Of Design Engineering," *Research-Technology Management*, 33(3), May 1990, pp. 23-27.

[2] Bentley, R., and Milburn, B., "Analysis of Pentium® Pro Processor Bugs," *Intel Design and Test Technology Conference*, June 1996. Intel Internal Document.

[3] Zucker, R., "Bug Root Cause Analysis for Willamette," *Intel Design and Test Technology Conference*, August 2000. Intel Internal Document.

AUTHORS' BIOGRAPHIES

Bob Bentley is the Pre-Silicon Validation Manager for the DPG CPU Design organization in Oregon. In his 17-year career at Intel he has worked on system performance evaluation, processor architecture, microprocessor logic validation and system validation in both pre- and post-silicon environments. He has a B.S. degree in Computer Science from the University of St. Andrews. His e-mail address is bob.bentley@intel.com

Rand Gray is a System Validation Manager for the DPG Validation organization in Oregon. He has more than 20 years of experience with microprocessors, including processor design, debug tool development, and systems validation. He has a B.S. degree in Computer Science from the University of the State of New York. His e-mail address is rand.gray@intel.com.